Speeding Up Linux Disk Encryption

Ignat Korchagin
@ignatkn
$ whoami

- Performance and security at Cloudflare
- Passionate about security and crypto
- Enjoy low level programming
Encrypting data at rest
The storage stack

applications
The storage stack

applications

filesystems
The storage stack

- applications
- filesystems
- block subsystem
The storage stack

- storage hardware
- block subsystem
- filesystems
- applications
Encryption at rest layers

- Applications
- Filesystems
- Block subsystem
- Storage hardware

SED, OPAL
Encryption at rest layers

- **Storage Hardware**
  - SED, OPAL

- **Block Subsystem**
  - LUKS/dm-crypt, BitLocker, FileVault

- **Filesystems**
  - BitLocker, FileVault

- **Applications**
  - BitLocker, FileVault
Encryption at rest layers

- **Storage hardware**: SED, OPAL
- **Block subsystem**: LUKS/dm-crypt, BitLocker, FileVault
- **Filesystems**: cryptfs, ext4 encryption or fscrypt
- **Applications**
Encryption at rest layers

- **Block subsystem**: LUKS/dm-crypt, BitLocker, FileVault
  - SED, OPAL
- **Filesystems**: LUKS/dm-crypt, BitLocker, FileVault
  - ecryptfs, ext4 encryption or fscrypt
- **Applications**: DBMS, PGP, OpenSSL, Themis

@ignatkn
Storage hardware encryption

Pros:

● it’s there
● little configuration needed
● fully transparent to applications
● usually faster than other layers
Storage hardware encryption

Pros:
● it’s there
● little configuration needed
● fully transparent to applications
● usually faster than other layers

Cons:
● no visibility into the implementation
● no auditability
● sometimes poor security

Block layer encryption

Pros:
- little configuration needed
- fully transparent to applications
- open, auditable
Block layer encryption

Pros:
- little configuration needed
- fully transparent to applications
- open, auditable

Cons:
- requires somewhat specialised crypto
- performance issues
- encryption keys in RAM
Filesystem layer encryption

Pros:

● somewhat transparent to applications
● open, auditable
● more fine-grained
● more choice of crypto + potential integrity support
Filesystem layer encryption

Pros:
● somewhat transparent to applications
● open, auditable
● more fine-grained
● more choice of crypto + potential integrity support

Cons:
● performance issues
● encryption keys in RAM
● complex configuration
● unencrypted metadata
Application layer encryption

Pros:

- open, auditable
- fine-grained
- full crypto flexibility
Application layer encryption

Pros:

- open, auditable
- fine-grained
- full crypto flexibility

Cons:

- encryption keys in RAM
- requires explicit support in code and configuration
- unencrypted metadata
- full crypto flexibility
Device mapper in Linux
Device mapper in Linux

filesystems
- ext4
- xfs
- vfat

file I/O

applications
Device mapper in Linux

filesystems
- ext4
- xfs
- vfat

file I/O

applications

block I/O

block device drivers
- scsi
- nvme
- brd
Device mapper in Linux

- **Filesystems**:
  - ext4
  - xfs
  - vfat

- **Applications**

- **Device Mapper**:
  - dm-raid
  - dm-crypt
  - dm-mirror

- **Block Device Drivers**:
  - scsi
  - nvme
  - brd

- **I/O Flows**:
  - File I/O
  - Block I/O
Device mapper in Linux

- Filesystems
  - ext4
  - xfs
  - vfat

- Device mapper
  - dm-raid
  - dm-crypt (highlighted)
  - dm-mirror

- Block device drivers
  - scsi
  - nvme
  - brd

- Applications

Connections:
- File I/O from filesystems to applications
- Block I/O from filesystems to device mapper, and from device mapper to block device drivers
dm-crypt (idealized)

- Filesystem
- Block device drivers
dm-crypt (idealized)

filesystem

dm-crypt

block device drivers
dm-crypt (idealized)

Filesystem → dm-crypt → Encrypt → Encrypted BIO → Block Device Drivers

write BIO
dm-crypt (idealized)

1. **filesystem**
   - Write BIO
   - Read BIO

2. **dm-crypt**
   - **encrypt**
   - **decrypt**

3. **encrypted BIOs**

4. **block device drivers**
dm-crypt (idealized)

- Files system
- write BIO
- read BIO
- dm-crypt
- encrypt
- decrypt
- encrypted BIOs
- block device drivers
- Linux Crypto API

@ignatkn
dm-crypt benchmarking
Test setup: RAM-based encrypted disk

$ sudo modprobe brd rd_nr=1 rd_size=4194304
Test setup: RAM-based encrypted disk

$ sudo modprobe brd rd_nr=1 rd_size=4194304
$ echo '0 8388608 delay /dev/ram0 0 0' |
sudo dmsetup create plain
Test setup: RAM-based encrypted disk

$ sudo modprobe brd rd_nr=1 rd_size=4194304
$ echo '0 8388608 delay /dev/ram0 0 0' | sudo dmsetup create plain
$ sudo cryptsetup luksFormat /dev/mapper/plain
Test setup: RAM-based encrypted disk

```
$ sudo modprobe brd rd_nr=1 rd_size=4194304
$ echo '0 8388608 delay /dev/ram0 0 0' | sudo dmsetup create plain
$ sudo cryptsetup luksFormat /dev/mapper/plain
$ sudo cryptsetup open --type luks /dev/mapper/plain secure
```
Test storage stack

- **secure**: dm-crypt
- **plain**: dm-delay
- **ramdisk**
Test storage stack

- secure: dm-crypt
- plain: dm-delay
- ramdisk

Optional (0 delay)
Test setup: sequential reads

$ cat rw.job

[iotest]
direct=1
gtod_reduce=1
loops=1000000
iodepth=16
Test setup: sequential reads

$ sudo fio --filename=/dev/mapper/plain
   --readwrite=read --bs=4k rw.job

... READ: io=21134MB, aggrb=1876.1MB/s
Test setup: sequential reads

$ sudo fio --filename=/dev/mapper/plain
   --readwrite=read --bs=4k rw.job

   READ: io=21134MB, aggrb=1876.1MB/s

$ sudo fio --filename=/dev/mapper/secure
   --readwrite=read --bs=4k rw.job

   READ: io=3261.8MB, aggrb=318.6MB/s
Test setup: sequential reads

$ sudo cryptsetup benchmark -c aes-xts

# Tests are approximate using memory only (no storage IO).

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Key</th>
<th>Encryption</th>
<th>Decryption</th>
</tr>
</thead>
<tbody>
<tr>
<td>aes-xts</td>
<td>256b</td>
<td>1854.7 MiB/s</td>
<td>1904.5 MiB/s</td>
</tr>
</tbody>
</table>
Test setup: sequential reads

```
$ sudo cryptsetup benchmark -c aes-xts
# Tests are approximate using memory only (no storage IO).
# Algorithm | Key | Encryption | Decryption
  aes-xts   256b | 1854.7 MiB/s | 1904.5 MiB/s
```

desired: ~900 MB/s, actual: ~300 MB/s
We tried...

- switching to different cryptographic algorithms
  - aes-xts seems to be the fastest (at least on x86)
We tried...

- switching to different cryptographic algorithms
  - aes-xts seems to be the fastest (at least on x86)
- experimenting with dm-crypt optional flags
  - "same_cpu_crypt" and "submit_from_crypt_cpus"
We tried...

- switching to different cryptographic algorithms
  - aes-xts seems to be the fastest (at least on x86)
- experimenting with dm-crypt optional flags
  - "same_cpu_crypt" and "submit_from_crypt_cpus"
- trying filesystem-level encryption
  - much slower and potentially less secure
Despair
Ask the community

“If the numbers disturb you, then this is from lack of understanding on your side. You are probably unaware that encryption is a heavy-weight operation...“

https://www.spinics.net/lists/dm-crypt/msg07516.html
But actually...

“Using TLS is very cheap, even at the scale of Cloudflare. Modern crypto is very fast, with AES-GCM and P256 being great examples.”

https://blog.cloudflare.com/how-expensive-is-crypto-anyway/
dm-crypt: life of an encrypted BIO request

filesystem

block device drivers

@ignatkn
dm-crypt: life of an encrypted BIO request

- filesystem
- dm-crypt
- Crypto API
- block device drivers
dm-crypt: life of an encrypted BIO request

- dm-crypt
- kcryptd
- filesystem
- Crypto API
- block device drivers
dm-crypt: life of an encrypted BIO request
dm-crypt: life of an encrypted BIO request

Diagram:
- dm-crypt
- kcryptd_io
- kcryptd
- Crypto API
- cryptd
- filesystem
- write
- block device drivers
dm-crypt: life of an encrypted BIO request

dm-crypt

kcryptd_io

write

Crypto API

cryptd

filesystem

block device drivers

dmcrypt_write

dm-crypt

cryptd

kcryptd

write
dm-crypt: life of an encrypted BIO request
dm-crypt: life of an encrypted BIO request
dm-crypt: life of an encrypted BIO request

@ignatkn
dm-crypt: life of an encrypted BIO request
dm-crypt: life of an encrypted BIO request
dm-crypt: life of an encrypted BIO request

- `dm-crypt`
- `kcryptd_io`
- `kcryptd`
- `dmcrypt_write`
- `Crypto API`
- `cryptd`
- `filesystem`
- `block device drivers`

@ignatkn
queues vs latency

“A significant amount of tail latency is due to queueing effects”

https://www.usenix.org/conference/srecon19asia/presentation/plenz
dm-crypt: life of an encrypted BIO request
dm-crypt: life of an encrypted BIO request
dm-crypt: git archeology

- kcryptd was there from the beginning (2005)
  - only for reads: “it would be very unwise to do decryption in an interrupt context”
dm-crypt: git archeology

● kcryptd was there from the beginning (2005)
  ○ only for reads: "it would be very unwise to do decryption in an interrupt context"

● some queuing was added to reduce kernel stack usage (2006)
dm-crypt: git archeology

- kcryptd was there from the beginning (2005)
  - only for reads: “it would be very unwise to do decryption in an interrupt context”
- some queuing was added to reduce kernel stack usage (2006)
- offload writes to thread and IO sorting (2015)
  - for spinning disks, but “may improve SSDs”
  - mentions CFQ scheduler, which is deprecated
dm-crypt: git archeology

- kcryptd was there from the beginning (2005)
  - only for reads: “it would be very unwise to do decryption in an interrupt context”
- some queuing was added to reduce kernel stack usage (2006)
- offload writes to thread and IO sorting (2015)
  - for spinning disks, but “may improve SSDs”
  - mentions CFQ scheduler, which is deprecated
- commits to optionally revert some queuing
  - “same_cpu_crypt” and “submit_from_crypt_cpus” option flags
dm-crypt: things to reconsider

- most code was added with spinning disks in mind
  - disk IO latency >> scheduling latency
dm-crypt: things to reconsider

- most code was added with spinning disks in mind
  - disk IO latency >> scheduling latency
- sorting BIOs in dm-crypt probably violates “do one thing and do it well” Unix principle
  - the task for the IO scheduler
dm-crypt: things to reconsider

● most code was added with spinning disks in mind  
  ○ disk IO latency >> scheduling latency
● sorting BIOs in dm-crypt probably violates “do one thing and do it well” Unix principle  
  ○ the task for the IO scheduler
● kcryptd may be redundant as modern Linux Crypto API is asynchronous by itself  
  ○ remove offloading the offload
dm-crypt: cleanup
dm-crypt: life of an encrypted BIO request
dm-crypt (synchronous)
dm-crypt (synchronous)
dm-crypt: removing queues

- dm-crypt module: a simple patch, which bypasses all queues/async threads based on a new runtime flag
dm-crypt: removing queues

● dm-crypt module: a simple patch, which bypasses all queues/async threads based on a new runtime flag

● Linux Crypto API is a bit more complicated
  ○ by default specific implementation is selected dynamically based on priority
  ○ aes-ni synchronous implementation is marked as “internal”
  ○ aes-ni (FPU) is not usable in some interrupt contexts
dm-crypt: removing queues

- dm-crypt module: a simple patch, which bypasses all queues/async threads based on a new runtime flag
- Linux Crypto API is a bit more complicated
  - by default specific implementation is selected dynamically based on priority
  - aes-ni synchronous implementation is marked as “internal”
  - aes-ni (FPU) is not usable in some interrupt contexts
- xtsproxy: a dedicated synchronous aes-xts module
xtsproxy crypto API module

aes-xts proxy
xtsproxy crypto API module

aes-xts proxy

Is FPU available?
xtsproxy crypto API module

Is FPU available?

yes

___xts-aes-aesni
xtsproxy crypto API module

aes-xts proxy

Is FPU available?

yes

__xts-aes-aesni

no

xts(ecb(aes-generic))
xtsproxy crypto API module

Is FPU available?

- __xts-aes-aesni
- xts(ecb(aes-generic))
Test setup: sequential IO

```bash
$ sudo fio --filename=/dev/mapper/secure
--readwrite=readwrite --bs=4k rw.job
```
Test setup: sequential IO

```
$ sudo fio --filename=/dev/mapper/secure
    --readwrite=readwrite --bs=4k rw.job
$ sudo modprobe xtsproxy
```
Test setup: sequential IO

```
$ sudo fio --filename=/dev/mapper/secure
  --readwrite=readwrite --bs=4k rw.job

$ sudo modprobe xtsproxy

$ sudo dmsetup table secure --showkeys | sed
  's/aes-xts-plain64/capi:xts-aes-xtsproxy-plain64/' | sed
  's/$/ 1 force_inline/' | sudo dmsetup reload
  secure
```
Test setup: sequential IO

```bash
$ sudo fio --filename=/dev/mapper/secure --readwrite=readwrite --bs=4k rw.job
$ sudo modprobe xtsproxy
$ sudo dmsetup table secure --showkeys | sed 's/aes-xts-plain64/capi:xts-aes-xtsproxy-plain64/' | sed 's/$/ 1 force_inline/' | sudo dmsetup reload secure
$ sudo dmsetup suspend secure && sudo dmsetup resume secure
```
ramdisk: read throughput
ramdisk: read throughput

inline enabled
ramdisk: write throughput
ramdisk: write throughput

inline enabled
ssd: IO latency (iowait)

- ssd disk
- dm-crypt device
ssd: IO latency (iowait)

- ssd disk
- dm-crypt device

inline enabled
Conclusions
Conclusions

● a simple patch which may improve dm-crypt performance by 200%-300%
  ○ fully compatible with stock Linux dm-crypt
  ○ can be enabled/disabled in runtime without service disruption
Conclusions

- a simple patch which may improve dm-crypt performance by 200%-300%
  - fully compatible with stock Linux dm-crypt
  - can be enabled/disabled in runtime without service disruption
- modern crypto is fast and cheap
  - performance degradation is likely elsewhere
Conclusions

● a simple patch which may improve dm-crypt performance by 200%-300%
  ○ fully compatible with stock Linux dm-crypt
  ○ can be enabled/disabled in runtime without service disruption

● modern crypto is fast and cheap
  ○ performance degradation is likely elsewhere

● extra queuing may be harmful on modern low latency storage
Caveats and future work

- the patch improves performance on small block size/high IOPS workloads
  - >2MB block size shows worse performance
Caveats and future work

- the patch improves performance on small block size/high IOPS workloads
  - >2MB block size shows worse performance
- the whole setup assumes hardware-accelerated crypto
  - xtsproxy supports x86 only
Caveats and future work

- the patch improves performance on small block size/high IOPS workloads
  - >2MB block size shows worse performance
- the whole setup assumes hardware-accelerated crypto
  - xtsproxy supports x86 only
- your mileage may vary
  - always measure and compare before deployment
  - let us know the results
Links

- https://gitlab.com/cryptsetup/cryptsetup
- https://github.com/cloudflare/linux
Questions?